Paper AA-13-2014

A SAS® Macro Using Parallel Genetic Algorithm to Automate Variable Selection

Jinqiao Li, MS., Alliance Data, Columbus, OH
Yi Cao, Ph.D., Alliance Data, Columbus, OH
Yanping Shen, MS., Alliance Data, Columbus, OH

1. ABSTRACT

Variable selection for generalized linear models has been a challenge in SAS® as current packages are either limited to a small scope of models or the searching algorithm still focuses on forward/backward/stepwise that biases parameter estimation ([1]).

The proposed macro VarSel_PGA was built upon such need. It utilizes the Genetic Algorithm (GA) as the searching algorithm. In contrast to traditional forward, backward and stepwise selection schemes, GA directs users to an optimal path to the final set of variables. When combined with PROC GENMOD, VarSel_PGA can be applied to any model in the procedure. To further enhance its performance, we incorporate p-value acceleration and parallelism in the macro. P-value acceleration technique facilitates algorithm to eliminate highly uninformative variables in the early stage of iterations thus expedites the convergence. Parallelism is used to aggregate the results from multiple paths of GA such that the consolidated result is more stable. In terms of fitness criteria, VarSel_PGA provides the flexibility for users to define the selection criteria (AIC, BIC) such that the final selection is tailored to their specific business needs.

In the paper, we will demonstrate variable selection results using this powerful macro with Logistic model and Gamma Generalized Linear Model. As GA based variable selection is highly expandable, extending the macro beyond the scope of models in GENMOD will be one of our future development directions.

2. INTRODUCTION

Variable selection has always been a challenging problem in model building process. Modelers usually have a large collection of variables to start with, and aim to build the ‘best’ model by selecting the most predictive variables from the collection.

In SAS®, several procedures have been developed to support the automatic variables selections for statistical models. For example, the GLMSELECT procedure has been used to select variables for general linear models in particular ([2]). However, this procedure does not support variable selection for the generalized linear models, which are the primary models in marketing domain when it comes to modeling customer purchasing amount, or a particular occurrence of a transaction.

A recently developed procedure HPGENSELECT available in SAS® 9.4 has extended to the generalized linear models ([3]). The main algorithms that implemented in the procedure are stepwise, forward and backward selections. However, the stepwise selection has been criticized for its greediness, which leads to the trap in local suboptimum ([4]). Alternative methods have been suggested to solve this problem, for example, an exhaustive search recommended by Miller ([4]), however, this is an expensive algorithm that clever algorithms are needed to lighten computation workload.

In this paper, we are interested in using a flexible algorithm, genetic algorithm, to solve the above issues. GA is a heuristic search tool originally designed for solving combinatorial optimization problems ([4]). To apply GA in statistical models, we developed a user-friendly macro - VarSel_PGA (variable selection using parallel genetic algorithm) in SAS. In this macro, two innovations were made to enhance the performance of standard GA. First, a p-value based acceleration technique greatly reduces the computational time. Second, instead of relying on single path evolution, parallel evolution is incorporated to integrate the results of multiple paths. We implemented VarSel_PGA on two datasets: simulated data and real data from a marketing company. In both cases, VarSel_PGA was proved to efficiently identify the predictive variables and hence provide the best solution to model building.

3. METHODOLOGY

In this section, we discuss three topics: single path genetic algorithm, a p-value based technique to accelerate the convergence and parallel genetic algorithm (PGA).

3.1 GENETIC ALGORITHM

Genetic algorithm is a stochastic searching algorithm. It can be applied to any problems formulated as function optimization. The logic of GA can be described by the flowchart in Figure 1.

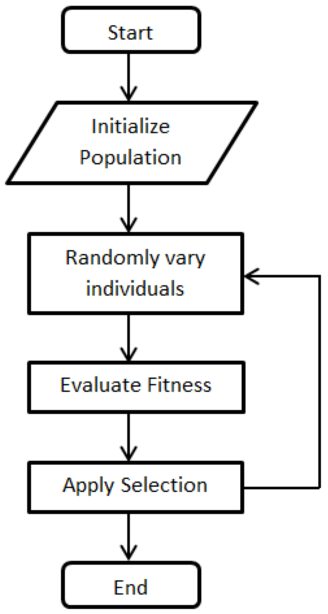


Figure 1: Flow chat of single path Genetic Algorithm

The idea of GA is derived from the Darwinian principle of “survival of the fittest”, the weaker individuals are gradually eliminated and the stronger ones survive and generate offspring. For the model selection purposes, a strong individual means it has good fitness score (lower AIC, BIC…), and a weak individual means it has bad fitness score (higher AIC, BIC etc.).

Before we dive into the details of the algorithm, some concepts need to be clarified first.

1. ***Individual:*** A group of variables are used as predictors of a model, represented by a binary string.
   For example, if there are 10 candidate predictors X_1, X_2, …, X_{10}, then individual {X_1, X_2, X_3} is represented by 1110000000; and individual {X_8, X_9, X_{10}} will be represented by 0000000111. Such a binary string can be viewed as DNA of an individual, and each bit can be viewed as a gene.

2. ***Individual size:*** The number of 1’s in an individual’s DNA.

3. ***Population:*** The complete set of individuals of a generation (iteration).

4. ***Population size:*** Number of individuals in a population, which is a pre-assigned and fixed even number.

We start our algorithm by randomly generating a population with population size *m*, individual length *l.* It takes four steps to generate a new generation: fitness statistics calculation, selection, reproduction, and mutation.

- Fitness statistics calculation: most SAS model fitting procedures report all kinds of fitness statistics, like AIC, AICC, BIC… For our macro, one can either choose AIC or BIC as the fitness statistic.

- Selection: after fitness statistics of all the individuals are derived, the stronger half of individuals is kept in the survival pool, and the weaker half is discarded.

- Reproduction: exchange the DNA fragment between two randomly selected cross-over positions of randomly selected two individuals from the survival pool as shown in Figure 2. Repeat this process *m/2* times, so that the new generation also has *m* individuals.

- Mutation: For each individual in the new generation, randomly select a mutation position and flip the gene of that position with probability *1/m.*

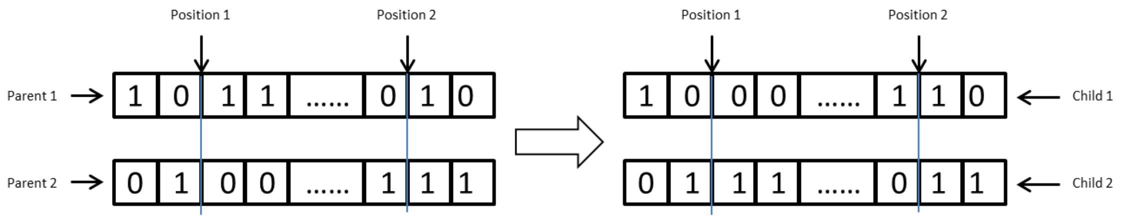


Figure 2: Crossover

For each generation, we go through above 4 steps and generate a new generation. This process stops when the convergence criteria is met, which is defined as *entropy* < δ (e.g. $\delta = 0.05$), a measure of disorder. To define entropy, let r_j be the frequency that the j^{th} gene is equal to 1 and the length of the DNA sequence is *p*. Then the entropy of a generation is given by Mu and Hugh (2006).

$$entropy = -\frac{1}{p}\sum_{j=1}^{p} r_j \ \log_2(r_j) + (1 - r_j)\ \log_2(1 - r_j)$$

When all the sequences in a population are identical, the entropy is 0 and we call the population is pure. However, in practice, it is not necessary to achieve 100% purity. When there are enough iteration, all the individuals in the last generation should be almost the same and one can choose the 'mode' of these individuals as the final selected model.

3.2 P-VALUE BASED ACCELERATION

The convergence of the algorithm becomes time consuming when the number of iterations is large. To accelerate, we propose to check the p-value of each variable for each individual after evaluating the individual. If a variable's p-value is larger than a pre-specified threshold, the variable is dropped immediately. P-value acceleration expedites the convergence by eliminating uninformative variable in early iteration.

The threshold can be set very aggressively, like 0.05, then the algorithm converges very fast and it also produces good results in some scenarios. However, it makes the algorithm resemble a combination of "backward selection" and GA. If one prefers keeping a pure GA flavor, a more tolerant threshold value, like 0.5, is recommended. The p-value criterion is only used to remove those highly uninformative variables in the first few generations so that the later generations can converge more quickly.

3.3 PARALLEL GENETIC ALGORITHM

Although GA sounds easy to understand and implement, practical experience shows that sometimes it is very hard to set all the control parameters correctly. In such scenarios, the algorithm tends to converge to suboptimal local solutions by randomness and the variable selection may not be consistent each run. To solve this issue, we introduce "majority vote" ([5]) concept to the process. That is, for some problems, aggregating information from several mediocre solutions may overcome randomness and produce a better solution. Parallel Genetic Algorithm is inspired by this very idea.

Realization of PGA is simple. To offset the influence of randomness, we run several paths of GA and count the appearance of each variable in the last generation for each path, and sum that across all the paths. Then we rank all the variables by their appearances in a frequency table. Variables appearing most frequently are those having the highest predictive power.

It is suggested that variables residing on the top of the frequency table should enter into consideration of the final model first. Mu and Hugh [6] suggest visually looking for a big gap between high frequency variables and low frequency variables, and use all the high frequency variables as the selected model. In practice, one may has some constrains on number of predictors or combination of predictors, so the final model is subjected to analyst's domain of knowledge, business feasibility or other concerns.

4. FUNCTIONALITY AND OUTPUT

VarSel_PGA, when packaged with PROC GENMOD, enables users to automate variable selection for most generalized linear models. A typical VarSel_PGA specification is as follows:

```
%VarSel_PGA (dsn=, y=, dist=, link=, n_indv=, indv_length=,
                          zero_vars =, zero_link =,
                          parallel =, sample_size =,
                          max_it=, stop_level=, kick_p=, Criteria =, exclude =);
```

Input	***Description***	***Default Value***
dsn	Data set name	NA
y	Response variable	NA
dist	Type of generalized linear models	NA
link	Link function of generalized linear models	NA
n_indv	Number of individuals.	NA
indv_length	Individual length. Number of variables in each individual	NA
zero_vars	Variables included in zero inflation part modeling	NA
zero_link	Link function for zero inflation part	NA
parallel	Number of parallels	10
sample_size	Sample size for each bootstrap	20,000
stop_level	Entropy threshold	0.1
kick_p	P value threshold	0.5
criteria	Model fitness criteria, AIC or BIC	NA
exclude	Variables that are not included in the variable selection step	NA

“dist” and “link” help specify primary models. When we need to model zero inflation, “zero_vars” and “zero_link” need to be specified as well. The specifications of “dist” are

Dist=	***Distribution***	***Link***
BINOMIAL | BIN | B	Binomial	Logit
GAMMA | GAM | G	Gamma	Inverse (power(−1))
GEOMETRIC | GEOM	Geometric	Log
IGAUSSIAN | IG	Inverse Gaussian	Inverse squared (power(−2))
MULTINOMIAL | MULT	Multinomial	Cumulative logit
NEGBIN | NB	Negative binomial	Log
NORMAL | NOR | N	Normal	Identity
POISSON | POI | P	Poisson	Log
ZIP	Zero-inflated Poisson	Log/logit
ZINB	Zero-inflated negative binomial	Log/logit

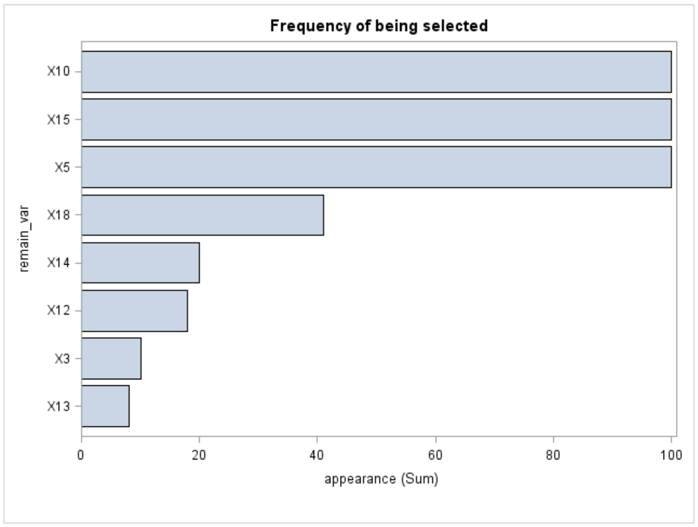
And the specifications of “link” are

Link=	***Description***
CUMCLL	
CCLL	Cumulative complementary log-log
CUMLOGIT	
CLOGIT	Cumulative logit
CUMPROBIT	
CPROBIT	Cumulative probit
CLOGLOG	
CLL	Complementary log-log
IDENTITY	
ID	Identity

LOG	Log
LOGIT	Logit
PROBIT	Probit
POWER(***number***) | POW(***number***)	Power with lambda= ***number***

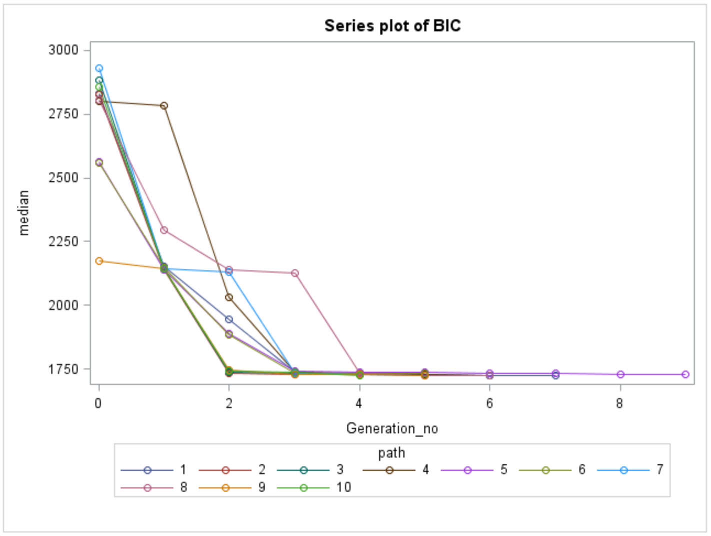
VarSel_PGA will generate four major outputs to assist users in determining in variable selection.

Output 1: "Frequency of being selected"



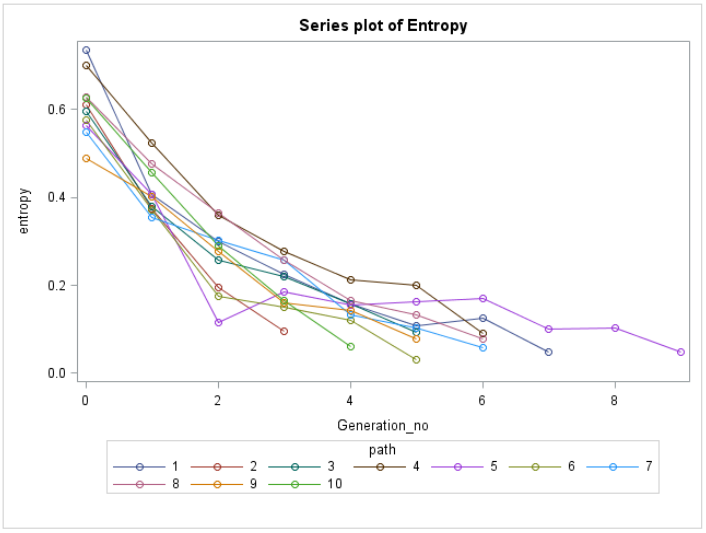
This graph outputs variables' predictive power from high to low. In practice, it is suggested that one should always consider the variables with high predictive power in the model first.

Output 2: "Series plot of BIC" or "Series plot of AIC"



Series plot of AIC/BIC demonstrate a series of median AIC/BICs of a population in each generation for each path. As less fitted individuals get eliminated during iterations, the median AIC/BICs tend to decrease.

Output 3: “Series plot of Entropy”



This plot provides a series of entropies of a population for each generation at each path. As we discussed before, small entropy (for example, 0.1) in the last generation means individuals are very similar across the population of the last generation. When entropy of last generation for a certain path is less than stop_level and the number of iterations has not reached the value specified in max_it, the iteration will stop, otherwise, iteration will continue until either entropy reaches stop_level or the number of iterations reaches max_it.

Output 4: The individual with the lowest BIC among all the paths will be outputted and computation time will be given

```
Converged !!!

        The following individual has the smallest BIC among all the paths

        X10
        X15
        X5


Plotting...


Finished !

this program used up 2 minutes and 52 seconds
```

An individual with the lowest BIC across the universe can be utilized as a candidate model. But we suggest users to cautiously select variables from the “Frequency of being selected” and modify the selection according to their own business needs, and compare with a candidate model. In order to come up with an optimal/meaningful model, some manual work is needed.

5. GUIDELINES

The variable selection process is completely automatic, as long as the model is specified and inputs are set in a sensible way. To make the macro more efficient, a few guidelines are provided:

- Theoretically, the more parallels users set, the more stable results are. However, it comes with longer computation time. Based on our experiments, the inputs’ impact on computation is:

  Number of parallels >> the number of individual > individual length

  To balance the impact of many parallels, we suggest assigning a relatively small number to sample size. For example, if you have 100,000 observations and need to parallel 20 times to get stable results. Choosing sample size = 10,000 with parallel = 20 is better than using all 100,000 observations for each parallel. Our experiments suggest parallel = 10 can be a good default setting.
- Individual length is preferable to be 1/3 to 1/2 of total number of initial variables
- Kick_p is not recommended to be too small (such as 0.05 or 0.1) as it will make entire algorithm lean towards backward selection

- Convergence is not a must for each path in parallel GA. By running a number of short paths that have not fully converged, spurious variables can be effectively eliminated. Therefore, when paralleled, unconvergence is not as severe issue as in single path GA

6. APPLICATIONS

6.1 SIMULATED DATA

Here is an example illustrating how to use VarSel_PGA to identify the predictive variables when building a logistic model. We first produced a simulated dataset consisting of 10,000 observations and 53 variables. Variables X_1, $X_{2...}$ X_{50} are continuous variables generated independently from uniform distributions. Variables C_1, C_2 and C_3 are categorical variables independent of each other. The model is:

$$logit(prob) = 2 + 5X_{17} - 8X_5 + 7I_{C_1=2} - 3I_{C_2=3} + 3X_2 + 7X_6 - 5X_3 + \varepsilon$$

where $\varepsilon \sim N(0,1)$ and Y=1 when $prob$ >0.5 and 0 otherwise. In other words, this example is constructed so that the dependent variable depends only on continuous variables (X_2, X_3, X_5, X_6, X_{17}) and classification variables (C_1, C_2). The following DATA step created the simulated data:

```
data logit_data;
       drop i j;
       array x{50} x1-x50;
              do i=1 to 10000;
                     do j=1 to 50;
                            x{j}=ranuni(1);
                     end;

                     c1=put(int(1.5+ranuni(1)*4),$2.);
                     c2=put(1+mod(i,3),$2.);
                     c3=put(int(ranuni(1)*9),$2.);

                     linpred=2+10*x17-8*x5+15*(c1=2)-19*(c2=3)+3*x2+7*x6-5*x3 +
                     rand("normal",0,1);
                     prob = exp(linpred)/ (1 + exp(linpred));
                     y = (prob > 0.5);
                     output;
              end;
       drop prob linpred;
run;
```

Then we run the macro VarSel_PGA to initiate variable selection, and the parameters setting are listed as follows:

```
%VarSel_PGA( dsn=logit_data, y=y, dist=b, link=logit, n_indv=50, indv_length=15,
             zero_vars = ,zero_link = ,parallel=20, sample_size = 1000,
             max_it=20, stop_level=0.1,kick_p=0.2, Criteria = BIC, exclude = );
```

A total 20 parallels are used, and each path evolves until stop_level reaches 0.1 or the number of iterations reaches 20. kick_p is set at 0.2 to exclude the insignificant variables at the early stage of iterations.

Figure 3 shows the results from VarSel_PGA. Variables are ranked based on their appearance frequency in the final paths, and only the top 15 variables are listed. From the plot, it shows that the seven variables used to generate theresponse variable are all on the top of the chart and well separated from the rest of irrelevant variables. The

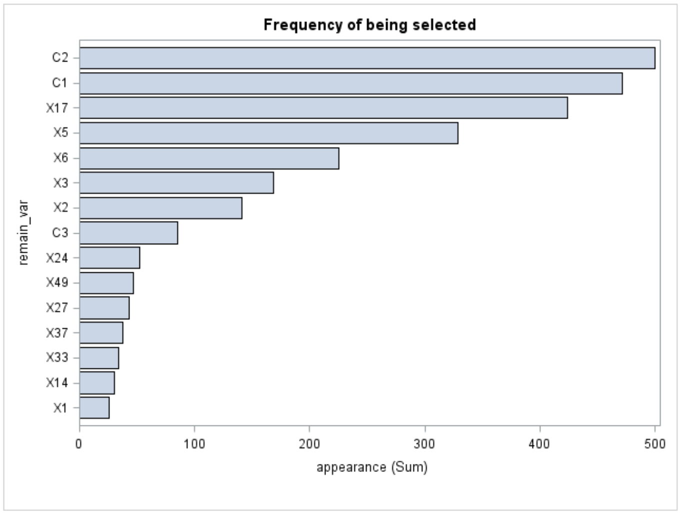


Figure 3: Frequency of being selected for simulated data

individual with the lowest BIC among all the paths is as below:

```
Converged !!!
          The following individual has the smallest BIC among all the paths
          C1
          C2
          X17
          X2
          X3
          X5
          X6

Plotting...

Finished !
```

Based on that, we conclude that the implementation of VarSel_PGA successfully find the correct model.

6.2 CUSTOMER FUTURE SPEND PREDICTION

Rewards, in addition to convenience, have become one of the main draws of credit card use. Nowadays, credit card companies rely on predictive models to project customer's future credit card spend and dig drivers of credit card usage such that they can customize rewards offers to drive incremental sales. In reality, with interactions of hundreds of variables, building a giant model consisting of all variables is feasible but unnecessary. In order to construct a concise and precise model, variable selection is strongly recommended. Figure 4 shows the factors that potentially impact a customer's future credit card usage.

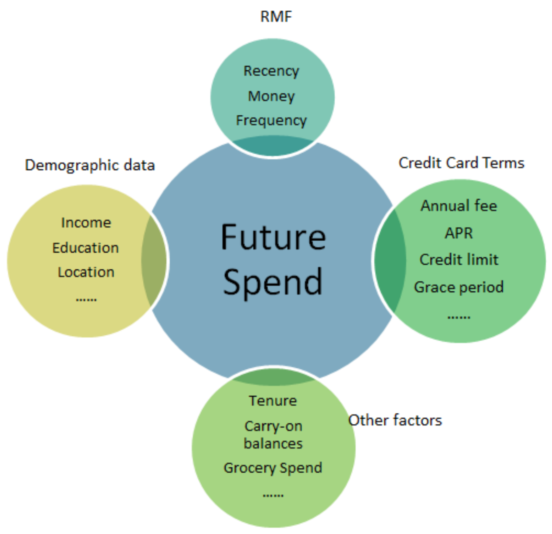


Figure 4: Factors that impact customers' future credit card usage

To frame the problem in statistics, let's assume a customer's future spend follows Gamma distribution, with other factors are linked with average future spend through logarithm. Below table lists the data content.

	Description
Y	Response variable – credit card spend in the next X months
# of numeric Xs	82
# of categorical Xs	2
# of observations	78,877

Following the guidelines, we choose 15 parallels and reduce sample size from default value 20,000 to 10,000. The number of individuals is set at 40 which is half of number of initial variables. Account ID is the distinct ID assigned to each customer and is irrelevant in prediction, so we exclude it in variable selection process by placing acct_id under "exclude".

```
%VarSel_PGA(dsn= data, y = net_sales_resp, dist= gamma, link=log,
n_indv=40, indv_length=30, sample_size = 10000, parallel= 15,
max_it=30, stop_level=0.1,kick_p=0.2, Criteria = BIC, exclude = acct_id);
```

After all iterations are over, we check the Entropy plot – Figure5. It turns out all paths are converged as entropies of last generation of all 15 paths are less than threshold 0.1.

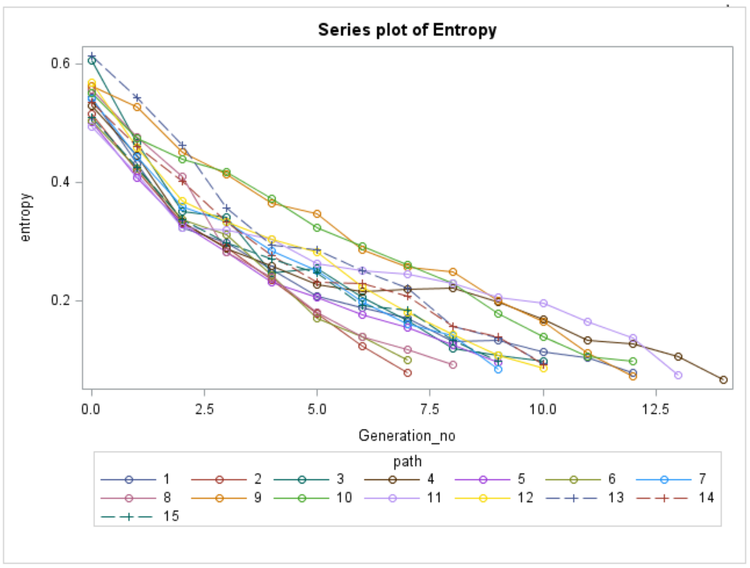


Figure 5: Series plot of Entropy

Then we check "Frequency of being selected"- Figure 6. Three big factors "RMF" (Recency, Net_Sales and Trip) are all selected and shown up on the position 1, 2 and 5 respectively. Other than that, some drivers are also found to be crucial, for example, X_{12} and X_4, as they trump Net_Sales to be the 3rd and 4th on Figure 6. For confidentiality, we mask all other factors as Xs.

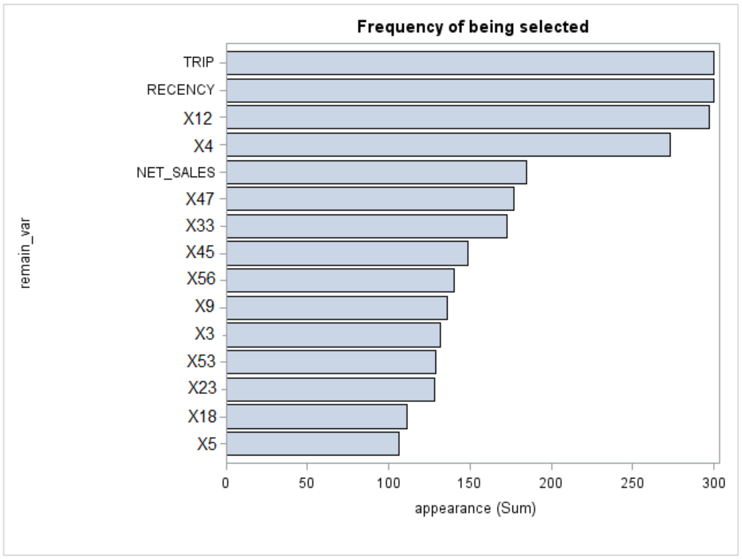
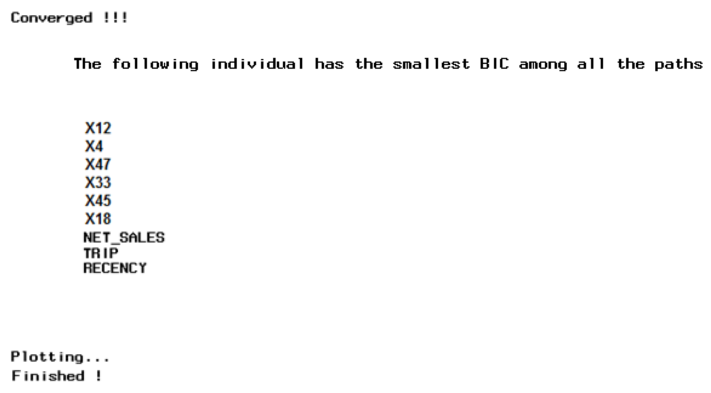


Figure 6: Frequency of being selected for Gamma generalized linear model

The macro will give an individual with the lowest BIC as below:



```
Converged !!!

        The following individual has the smallest BIC among all the paths


          X12
          X4
          X47
          X33
          X45
          X18
          NET_SALES
          TRIP
          RECENCY



Plotting...
Finished !
```

Even though a model with the lowest BIC across the universe is given, we are not indicating that modeling process stops at this point. Instead, we suggest users to fully leverage the “Frequency of being selected” plot and tailor the final model to specific business needs.

7. CONCLUSIONS

VarSel_PGA allows users to automate variable selection process in a great extent. It utilizes Genetic Algorithm in machine learning domain to optimize the search path to the final set of predictive variables. In addition, by aggregating results from multiple paths of GA, the macro successfully reduces the variation introduced by single path of GA and the result is expected to be more reliable. The performance of parallel Genetic Algorithm is further improved by p-value based acceleration technique as uninformative variables are eliminated in the early iterations.

VarSel_PGA is an extremely flexible variable selection tool in that the algorithm is completely independent of model, which makes the macro applicable to any model as long as model’s p value and fitness criteria can be captured. In current implementation, the macro is wrapped with PROC GENMOD as PROC GENMOD plays a role of model generator. To take full advantage of such flexibility, our next steps dwell on incorporating models that have not yet been covered by PROC GENMOD, for example, survival model and tweedie model.

REFERENCES

[1] Mark J. Whittingham, Philip A. Stephens. Why do we still use stepwise modelling in ecology and behavior? Journal of Animal Ecology. 2006, 75, 1182-1189

[2] Robert A. Cohen. “Introducing the GLMSELECT PROCEDURE for Model Selection,” SAS Institute Inc. Cary, NC. 2006

[3] Cohen, R. and Rodriguez, R. N. High-performance statistical modeling. Proceedings of the SAS Global Forum 2013 Conference, Cary, NC: SAS Institute Inc. 2013

[4] Goldberg, D. E. Genetic Algorithms in Search, Optimization and Machine Learning, Reading, MA: Addison-Wesley. 1989

[5] James, G. "Majority-Vote Classifiers: Theory and Applications," unpublished doctoral dissertation, Stanford University, Department of Statistics.

[6] Mu Zhu, Hugh A. Chipman. Darwinian Evolution in Parallel Universes: A Parallel Genetic Algorithm for Variable Selection. Technometrics, November, 2006 Vol 48. No 4.

ACKNOWLEDGMENTS

We would like to profusely thank Ning Ma, Amy Yu and Tim Sweeney for their guide and mentorship during the course of this work. We also thank Marketing Insights Department at Alliance Data for providing us the opportunity to attend the conference.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Name: Jinqiao Li
Enterprise: Alliance Data
Address: 3100 Easton Square Pl
City, State, ZIP: Columbus, Ohio, 43219
Work Phone: 614-944-3467
E-mail: jinqiao.li@alliancedata.com

Name: Yi Cao
Enterprise: Alliance Data
Address: 3100 Easton Square Pl
City, State, ZIP: Columbus, Ohio, 43219
Work Phone: 614-944-3716
E-mail: yi.cao@alliancedata.com

Name: Yanping Shen
Enterprise: Alliance Data
Address: 3100 Easton Square Pl
City, State, ZIP: Columbus, Ohio, 43219
Work Phone: 614-944-3685
E-mail: yanping.shen@alliancedata.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

APPENDIX

Below is SAS code for macro %VarSel_PGA

```
%macro VarSel_PGA(dsn, y, dist, link, n_indv, indv_length, zero_vars = ,
               zero_link = log, parallel=10, sample_size = 20000, max_it=20,
stop_level=0.1,kick_p=0.5, Criteria = BIC, Exclude = 0);

options nonotes nosource nosource2 errors=0;
ods listing close;
ods results off;

%let time_start = %sysfunc(DATETIME());

        proc datasets lib=work NOLIST;
                delete _output_entropy _output_individuals_all _output_&Criteria;
        run;
        quit;
        %let y = %sysfunc(upcase(&y));
        %let dsn = %sysfunc(upcase(&dsn));
        %let dist = %sysfunc(upcase(&dist));
        %let Criteria = %sysfunc(upcase(&Criteria));
        %let Exclude = &y %sysfunc(upcase(&Exclude));

        data _null_;
                A = "&Exclude";
                num_word = countw(A);
                length quote_A $1000.;
                do i = 1 to num_word;
                        word = upcase(cats("'",scan(A,i),"'"));
                        quote_A = catx(',',quote_A,word);
                end;
                Call symputx('quoted_Exclude',quote_A);
        run;

        %let dsid=%sysfunc(open(&dsn));
        %let num=%sysfunc(attrn(&dsid,nlobs));
        %let rc=%sysfunc(close(&dsid));

        %do i = 1 %to 30; /* Print some blank lines, clean log */
                %put;
        %end;

        %if &num > &sample_size %then %do;
                %put Large data, sampling needed, sample size is &sample_size;
                %let sample_ind = 1;
        %end;

        %else %do;
                %put This data set is not very large, no sampling needed;
                %let sample_ind = 0;
        %end;

        proc sql noprint;
        /* Get a list of predictors */
                select upcase(name) into :x_list separated by '|'
                        from dictionary.columns
                        where libname='WORK'
                                   and memname="&dsn"
                                   and upcase(name) not in (&quoted_Exclude);
        quit;
        %let num_vars = &SQLOBS;
        %put there are &num_vars non-response variables in &dsn;

        proc sql noprint;
        /* Get a list of character variables in &x_list */
                select upcase(name) into :char_x_list separated by '|'
                        from dictionary.columns
                        where libname='WORK'
                                   and memname="&dsn"
```

```
                                  and upcase(name) not in (&quoted_Exclude)
                                  and type = 'char';

        quit;
        %let num_chars = &SQLOBS;
        %put there are &num_chars character variables in &dsn;

        %if not %symexist(char_x_list) %then %do;
                %let char_x_list = ;
        %end;

        %if &zero_vars ne %then %do;
                %let zero_vars = %sysfunc(upcase(&zero_vars));
                /* Get a list of character variables in &zero_vars */
                data _null_;
                        A = "&zero_vars";
                        num_word = countw(A);
                        length quote_A $1000.;
                        do i = 1 to num_word;
                                word = upcase(cats("'",scan(A,i),"'"));
                                quote_A = catx(',',quote_A,word);
                        end;
                        Call symputx('quoted_zero_vars',quote_A);
                run;

                proc sql noprint;
                        select upcase(name) into :zero_chars separated by ' '
                                from dictionary.columns
                                where libname='WORK'
                                          and memname="&dsn"
                                          and upcase(name) in (&quoted_zero_vars)
                                          and type = 'char';
                quit;
        %end;

        %do path = 1 %to &parallel;
                %let Generation_no = 0;
                %let entropy = 1;
                %let model_data = &dsn;

                %if &sample_ind = 1 %then %do;
                %let model_data = sample_data;
                        %put Sampling... for path &path,sample_size is &sample_size;
                        proc surveyselect data=&dsn noprint
                                                  method=srs
                                                  out=&model_data
                                                  n=&sample_size;
                        run;
                %end;

                %gen_indiv(num_vars=&num_vars,l=&indv_length,n=&n_indv);

                %do %while ((&Generation_no <= &max_it) and (&entropy >= &stop_level));
                        data Individuals_2;
                                Path = &path;
                                Generation_no = &Generation_no ;
                                id = _n_;

                                set Individuals;
                                length var_list $20000. char_list $15000. vars_indv $2000.
indv_chars $1500.;
                                retain var_list char_list;

                                if _n_ = 1 then do;
                                        var_list = SYMGET('x_list');
                                        char_list = SYMGET('char_x_list');
                                        call missing(vars_indv,indv_chars);
                                end;

                                indv_len = 0;
                                do i = 1 to &num_vars;
```

```
                    /* For each individual, parse its '01' binary code; and save
                       the parsed variable list in macro variables */
                            _in_out_ind_ = char(individual,i);
                            if _in_out_ind_ = '1' then do;
                                    _vvv_ = scan(var_list,i,'|');
                                    vars_indv = catx(' ',vars_indv,_vvv_);
                                    indv_len + _in_out_ind_;
                                    do j = 1 to &num_chars;
                                            if _vvv_ = scan(char_list,j,'|') then
indv_chars = catx(' ',indv_chars,_vvv_);
                                    end;
                            end;
                    end;
                    call symputx(cats('indv_',id),vars_indv);
                    call symputx(cats('indv_char_',id),indv_chars);

                    drop _in_out_ind_ _vvv_ i j var_list char_list;
            run;

            /* Fit models */

            proc datasets lib=work NOLIST;
                    delete _fit_stats_all _par_ests_all;
            run;
            quit;
            %do i = 1 %to &n_indv;
            /* Fit each individual of this generation, and keep their stats */

                    %if (&dist = ZIP | &dist = ZINB) %then %do;
                            proc genmod data=&model_data  namelen=32;
                                    class &char_x_list;
                                    model &y = &&INDV_&i/dist=&dist link=&link ;
                                    zeromodel  &zero_vars /link = &zero_link ;
                                    ods output ModelFit   = _fit_stats;
                                    ods output ParameterEstimates = _par_ests;
                            run;
                    %end;

                    %else %do;
                            proc genmod data=&model_data  namelen=32;
                                    class &&INDV_CHAR_&i;
                                    model &y = &&INDV_&i/dist=&dist link=&link ;
                                    ods output ModelFit   = _fit_stats;
                                    ods output ParameterEstimates = _par_ests;
                            run;
                    %end;

                    %put Now we are at Path &path, Generation &Generation_no,
individual &i;

                    data _fit_stats;
                            id = &i;
                            set _fit_stats;
                    run;

                    data _par_ests;
                            path = &path;
                            id = &i;
                            length Parameter $40.;
                            set _par_ests;
                            keep Parameter ProbChiSq id path;
                    run;

                    proc append base=_fit_stats_all data=_fit_stats force;
                    run;

                    proc append base=_par_ests_all data=_par_ests force;
                    run;
            %end;

            proc transpose data=_fit_stats_all
```

```
                                                out=_fit_stats_all_2(drop=_NAME_

        rename=(AIC__smaller_is_better_ = AIC

        BIC__smaller_is_better_ = BIC));
                              by id;
                              var value;
                              id Criterion;
                      run;

                      /* Keep track of the generation's median AIC or BIC */
                      proc means data=_fit_stats_all_2;
                              output out=Criteria_median(drop= _type_ _freq_)
median(&Criteria)=median;
                      run;

                      data Criteria_median;
                              set Criteria_median;
                              path = &path;
                              Generation_no = &Generation_no ;
                              call symputx('median',median);
                      run;

                      proc append base=_output_&Criteria data=Criteria_median;
                      run;


                      /*      Drop half large AIC/BIC individuals   */
                      proc sort data=_fit_stats_all_2; by &Criteria ; run;

                      data _fit_stats_all_2;
                              set _fit_stats_all_2;
                              path = &path;
                              Generation_no = &Generation_no;
                              if _n_ <= &n_indv / 2 then keep_ind = 1;
                              else keep_ind = 0;
                              keep id &&Criteria path Generation_no keep_ind;
                      run;

                      proc sql noprint;
                              select id into :selected_ID separated by ','
                              from _fit_stats_all_2
                              where keep_ind = 1;
                      quit;

                      proc append base=_indv_&Criteria data=_fit_stats_all_2;
                      run;
                      /* Modify each individual by kicking out those large p-value variables */
                      proc sql noprint;
                              create table _par_ests_all_2 as
                              select id,upcase(Parameter) as Parameter,min(ProbChiSq) as p_value
                              from _par_ests_all
                              where Parameter ne 'Intercept'
                                       and id in (&selected_ID)
                              group by id,Parameter
                              order by id;
                      quit;

                      data _par_ests_all_3;
                              retain path Generation_no id;
                              set _par_ests_all_2;
                              length vars_remain $2000. var_list $20000. indiv_2 $&num_vars..;
                              if _n_ = 1 then do;
                                      var_list = SYMGET('x_list');
                              end;
                              by id;
                              retain vars_remain var_list;
                              if first.id then do;
                                      call missing(vars_remain);
                                      num_remain = 0;
                              end;
```

```
                if p_value < &kick_p then do;
                        vars_remain = catx('|',vars_remain,Parameter);
                        num_remain + 1;
                end;
                if last.id then do;
                        indiv_2 = repeat('0',&num_vars-1);
                        Generation_no = &Generation_no;
                        path = &path;
                        do i = 1 to &num_vars;
                                v = scan(var_list,i,'|');
                                do j = 1 to num_remain;
                                        u = scan(vars_remain,j,'|');
                                        if u = v then substr(indiv_2,i,1)=1;
                                end;
                        end;
                        output;
                end;
                keep path Generation_no id vars_remain indiv_2;
        run;

        proc append data=_par_ests_all_3 base=_output_individuals_all force;
        run;

        /* Pre-mutate: select parents */
        proc surveyselect data=_par_ests_all_3 noprint
                                method=srs
                                out=temp_keep_indiv
                                n=2
                                rep = %eval(&n_indv/2);
        run;

        proc transpose data=temp_keep_indiv out=mutate_1 prefix=indv_;
                by Replicate;
                var indiv_2;
        run;

        /* Cross-over & Mutate*/
        data mutate_1;
                set mutate_1;
                pos1 = ceil((&num_vars-1)*ranuni(0));
                pos2 = ceil((&num_vars-2)*ranuni(0));
                call sortn(pos1,pos2);
                after_exch_indv_1 = indv_1;
                after_exch_indv_2 = indv_2;
                gen_sub = substr(after_exch_indv_1,pos1+1,pos2-pos1);
                substr(after_exch_indv_1,pos1+1,pos2-pos1) =
substr(after_exch_indv_2,pos1+1,pos2-pos1);
                substr(after_exch_indv_2,pos1+1,pos2-pos1) = gen_sub;
                mut_pos_1 = ceil(&num_vars*ranuni(0));
                mut_pos_2 = ceil(&num_vars*ranuni(0));
                after_mut_indv_1 = after_exch_indv_1;
                after_mut_indv_2 = after_exch_indv_2;

/*              if ranuni(0) < (1/2) then do;*/
                if ranuni(0) < (1/&num_vars) then do;
                        substr(after_mut_indv_1,mut_pos_1,1) = not
char(after_exch_indv_1,mut_pos_1);
                end;

                if ranuni(1) < (1/&num_vars) then do;
                        substr(after_mut_indv_2,mut_pos_2,1) = not
char(after_exch_indv_2,mut_pos_2);
                end;
                drop _NAME_;
        run;

        data Individuals_mutate_1;
                set Mutate_1;
                by Replicate;
                mutate_indv = after_mut_indv_1;
                output;
```

```
                                mutate_indv = after_mut_indv_2;
                                output;
                                keep Replicate mutate_indv;
                        run;

                        data Individuals;
                                set Individuals_mutate_1;
                                id = _n_;
                                rename mutate_indv = Individual ;
                        run;

                        /* Calculate entropy */
                        %calculate_entropy(dsn=Individuals,n=&num_vars);
                        data Entropy_out;
                                set Entropy_out;
                                path = &path;
                                Generation_no = &Generation_no;
                                call symputx('entropy',entropy);
                        run;
                        proc append data=entropy_out base=_output_entropy;
                        run;

                        %put;
                        %put %str(Current generation%'s entropy is &entropy, median &Criteria is
&median);
                        %put;

                        %if &entropy < &stop_level %then %do;
                                %put;
                                %put Converged !!!;
                                %put;
                        %end;

                        %if (&Generation_no = &max_it & &entropy > &stop_level) %then %do;
                                %put;
                                %put Not converge yet, but max iteration level reached proceed
anyway...;
                                %put;
                        %end;

                        %let Generation_no = %eval(&Generation_no + 1);
                %end;
        %end;

        /* Prepare for frequency bar chart, for each path only using the last generation */
        proc sort data=_output_individuals_all out=_count_freq_1;
                by path descending Generation_no;
        run;

        data _count_freq_2; /* only keep last generation */
                retain keep_ind;
                set _count_freq_1;
                by path descending Generation_no;
                lag_Generation_no = lag(Generation_no);
                if first.Generation_no then keep_ind = 1;
                if lag_Generation_no > Generation_no then keep_ind = 0;
                if keep_ind;
                keep path Generation_no id vars_remain;
        run;

        data _count_freq_3;
                set _count_freq_2 end = eof;
                indv_len = countw(vars_remain,'|');
                do i = 1 to indv_len;
                        remain_var = scan(vars_remain,i);
                        if remain_var not in ('SCALE','DISPERSION') then output;
                end;
        run;

        proc sql noprint;
                create table _freq_result as
                select remain_var, count(*) as appearance
                from _count_freq_3
                group by remain_var
                order by appearance desc,remain_var ascending;
        quit;

        data _freq_result;
                set _freq_result;
                if _n_ <= 15;
        run;

        proc sql noprint;
                create table _top_1 as
                select vars_remain, &Criteria
                from _count_freq_2 A join _indv_&Criteria B
                        on (A.path = B.path
                           and A.Generation_no = B.Generation_no
                           and A.id = B.id)
                order by &Criteria;
        quit;

        data _null_;
                set _top_1(obs=1);
                num = countw(vars_remain);
                putlog "         The following individual has the smallest &Criteria among all the
paths ";
                putlog ;
                do i = 1 to num;
                        a = scan(vars_remain,i);
                        if a not in  ('SCALE','DISPERSION') then
                                putlog '         ' a;
                end;
                putlog;

        run;

        %let timenow=%sysfunc(time(), hhmm.);
        %let time_compress = %sysfunc(compress(&timenow,':'));
        %let datenow=%sysfunc(date(), date9.);

        %put;%put;
        %put Plotting...;

        ods results;
        /* Below are the result path, please change accordingly */
        ods html
file="C:\GA_results\results_&datenow._&time_compress\results_&datenow._&time_compress..html"
                         gpath="C:\GA_results\results_&datenow._&time_compress\images";
        ods graphics on / imagename="GA_plot";

        footnote "This file is saved at C:\GA_results\results_&datenow._&time_compress";
        proc sgplot data=_freq_result;
                title 'Frequency of being selected';
                hbar remain_var / response=appearance;
                yaxis discreteorder=data;
        run;

        proc sgplot data=_output_&Criteria;
                title "Series plot of &Criteria";
                series x = Generation_no y= median / group=path markers;
        run;

        proc sgplot data=_output_entropy;
                title "Series plot of Entropy";
                series x = Generation_no y= Entropy / group=path markers;
        run;
        footnote '';

        ods html close;
        ods results off;
```

```
       proc datasets lib=work NOLIST;
               delete Entropy_out Individuals Individuals_2 Individuals_all_temp
Individuals_mutate_1
                         Mutate_1 Temp_keep_indiv _fit_stats_all _fit_stats_all_2 _par_ests_all
_par_ests_all_2
                         _par_ests_all_3 _fit_stats _par_ests Criteria_median _count_freq_1
_count_freq_2
                         _count_freq_3 sample_data _top_1 _top_2 _indv_&Criteria;
       run;
       quit;

%put;
%put Finished !;
%put;

data _null_;
       seconds = intck('second',&time_start,DATETIME());
       call symputx('mm',floor(seconds/60));
       call symputx('ss',mod(seconds, 60));
run;

%put this program used up &mm minutes and &ss seconds;
ods listing;
options notes source source2 errors=20;
%mend;
```